RAY TRACING RENDER MENGGUNAKAN *FRAGMENT ANTI ALIASING*

Febriliyan Samopa[1] Wawan Mardian[1]

[1]Jurusan Teknik Informatika, Fakultas Teknologi Informasi, Institut Teknologi Sepuluh Nopember

ABSTRACT

Rendering is generating surface and three-dimensional effects on an object displayed on a monitor screen. Ray tracing as a rendering method that traces ray for each image pixel has a drawback, that is, aliasing (jaggies effect). There are some methods for executing anti aliasing. One of those methods is OGSS (Ordered Grid Super Sampling). OGSS is able to perform aliasing well. However, this method requires more computation time since sampling of all pixels in the image will be increased. Fragment Anti Aliasing (FAA) is a new alternative method that can cope with the drawback. FAA will check the image when performing rendering to a scene. Jaggies effect is only happened at curve and gradient object. Therefore, only this part of object that will experience sampling magnification. After this sampling magnification and the pixel values are computed, then downsample is performed to retrieve the original pixel values. Experimental results show that the software can implement ray tracing well in order to form images, and it can implement FAA and OGSS technique to perform anti aliasing. In general, rendering using FAA is faster than using OGSS although there is some situation where the time effectiveness is equal. This is possible since the process of intersection searching on FAA needs additional time. Rendered images using FAA are relatively similar to those using OGSS.

Keywords: *ray tracing, fragment anti aliasing, ordered grid super sampling, render, sampling, jaggies*

ABSTRAK

Render adalah pemberian permukaan dan kesan tiga dimensi pada objek yang ditampilkan pada layar monitor. Ray Tracing atau penulusuran sinar sebagai suatu metode render yang menelusuri sinar untuk tiap piksel pada citra adegan mempunyai kelemahan yaitu timbulnya alias (efek jaggies atau efek tangga). Terdapat beberapa metode yang digunakan untuk melakukan anti aliasing diantaranya OGSS (Ordered Grid Super Sampling). OGSS mampu melakukan aliasing dengan baik. Hanya saja metode ini memakan waktu yang lebih banyak karena semua piksel dalam citra akan diperbesar samplingnya. Fragment Anti Aliasing (FAA) sebagai metode baru merupakan alternatif untuk mengatasi kekurangan tersebut. FAA akan melakukan pengecekan terhadap citra saat melakukan render terhadap adegan. Efek jaggies hanya terjadi pada objek yang berupa lengkungan dan kemiringan, sehingga hanya bahian ini saja dari citra yang akan mengalami proses perbesaran sampling. Setelah sampling diperbesar dan didapatkan nilai piksel, maka akan dilakukan downsample untuk mendapatkan nilai warna piksel asli. Dari hasil ujicoba didapatkan bahwa perangkat lunak dapat menerapkan penelusuran sinar dengan baik untuk membentuk citra, serta dapat menerapkan teknik FAA serta OGSS untuk melakukan anti aliasing. Pada umumnya render dengan FAA lebih cepat dibandingkan dengan OGSS, meskipun terdapat keadaan tertentu dimana keefektifan waktunya sama. Hal ini dimungkinkan karena untuk proses pencarian perpotongan pada FAA membutuhkan waktu tersendiri. Citra hasil render dengan menerapkan FAA relatif sama dengan citra hasil render menerapkan OGSS.

Kata Kunci: *ray tracing, fragment anti aliasing, ordered grid super sampling, render, sampling, jaggies, penelusuran sinar*

Render adalah pemberian permukaan dan kesan tiga dimensi pada obyek yang ditampilkan pada layar monitor [1]. *Ray Tracing* atau penelusuran sinar sebagai suatu metode *render* yang banyak digunakan, mempunyai kelemahan yaitu timbulnya *alias* atau adanya efek *jaggies* atau efek tangga pada tepian obyek yang berbentuk lengkungan dan tepian obyek yang mempunyai kemiringan. Terdapat beberapa metode yang digunakan untuk melakukan anti *aliasing* diantaranya OGSS (*Ordered Grid Super Sampling*).

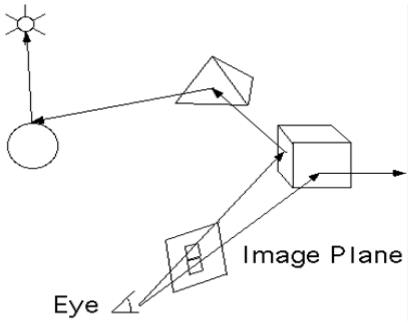
OGSS mampu melakukan *aliasing* dengan baik. Hanya saja metode ini memakan waktu yang lebih lama. Hal ini disebabkan tidak adanya pengecekkan terhadap gambar yang akan dilakukan proses anti *aliasing*. Oleh karena itu, pada garis lurus vertikal dan horizontal pun akan dilakukan proses anti *aliasing*. Padahal efek *jaggies* hanya terjadi pada lengkungan dan kemiringan.

Fragment Anti Aliasing (FAA), merupakan salah satu metode baru dalam proses penghilangan *jaggies* pada gambar. FAA dikembangkan sebagai alternatif selain OGSS untuk mengatasi kelemahan dari *Ray Tracing* [2]. Penerapan FAA diharapkan dapat memperbaiki citra hasil *render* dan mengatasi kelemahan pada metode sebelumnya.

RAY TRACING

Ray Tracing adalah salah satu teknik *render* yang masukannya adalah deskripsi dari ruang tiga dimensi untuk di-*render*. Deskripsi tersebut terdiri dari lokasi kamera, arah pandang kamera, posisi cahaya, serta data-data yang berkaitan dengan sumber cahaya dan data deskripsi benda yang ada dalam ruang tiga dimensi itu sendiri [3].

Cara kerja *Ray Tracing* adalah sama dengan kejadian nyata, yakni permukaan obyek yang di-*render* tergantung dari sinar yang mencapai mata kita. Dalam hal ini, sifat sinar dimodelkan dalam program yang me-*render* sebuah



Gambar 1: Penelusuran dari mata.

adegan. Dalam kejadian nyata, cahaya atau sinar berasal dari satu atau lebih sumber, yang kemudian mengenai benda atau obyek, memantul dan kadang mencapai mata pengamat (Gambar 1).

Ketika sinar memancar dan mengenai sebuah obyek, energinya akan diserap dan sebagian dipantulkan. Sinar yang dipantulkan membawa warna dari obyek [4]. Sinar yang memantul bisa mengenai obyek lain dan mempengaruhi warna obyek tersebut, atau tidak mengenai obyek sama sekali dan bisa juga mencapai mata pengamat sehinggga pengamat dapat melihat obyek.

Untuk dapat me-*render* obyek, baik obyek geometris maupun obyek non geometris (obyek yang dibentuk dari jaringan poligon) maka syarat yang harus dipenuhi ialah program mampu mencari perpotongan antara sinar dan obyek, selain itu juga dapat memperoleh vektor normal dari obyek tersebut. Dalam penelitian ini, obyek yang ditampilkan adalah obyek geometris saja, karena obyek geometris lebih mudah ditampilkan dan tidak perlu membuat jaringan poligonnya terlebih dahulu.

Ray Tracing bekerja dengan cara menentukan bidang pandang dan arah pandang terlebih dahulu. Untuk setiap piksel pada bidang pandang (bidang pandang merepresentasikan citra yang akan dihasilkan), ditembakkan satu sinar dengan titik awal sinar pada mata, dan titik akhir sinar pada titik yang diwakili oleh piksel tersebut. Untuk tiap obyek pada adegan, dicari titik perpotongan terdekat dengan sinar tersebut. Apabila ditemukan, maka diterapkanlah model pencahayaan pada obyek ini untuk menentukan warna piksel tadi [5].

Ray Tracing adalah algoritma iluminasi global. Iluminasi global dalam grafika komputer adalah istilah yang diberikan untuk model-model untuk me-*render* suatu adegan dengan mengevaluasi cahaya yang dipantulkan dari sebuah titik x dengan membawa semua iluminasi yang sampai ke titik tersebut. Hal ini berarti nilai cahaya yang diperhitungkan bukan hanya dari sumber cahaya secara langsung, tetapi dari semua iluminasi tak langsung yang mungkin berasal dari sumber cahaya lewat obyek lain.

Ray Tracing menggabungkan beberapa elemen berikut ke dalam suatu model tunggal: a) Penghilang permukaan yang tersembunyi (*hidden -surface removal*); b) Pemberian warna akibat iluminasi langsung; c) Efek interaksi spekular umum seperti refleksi dan refraksi; d) Komputasi bayangan; e) Proyeksi perspektif.

Penghilangan permukaan yang tersembunyi (yang tidak terlihat oleh mata) secara otomatis dilakukan oleh *Ray Tracing*, karena sinar yang ditelusuri dari mata akan digunakan untuk menentukan titik perpotongan sinar itu dengan obyek. Titik perpotongan yang akan diambil adalah yang terdekat dengan mata. Pemberian warna akibat model iluminasi akan dibahas pada sub bab yang lain. Efek refleksi (pantulan) dan refraksi (pembiasan) dapat diimplementasikan karena dengan representasi sinar dalam bentuk vektor dapat juga dicari sinar pantul dan sinar bias juga dalam bentuk vektor. Untuk menentukan warna yang dibawa oleh sinar pantul dan sinar bias ini, algoritma *Ray Tracing* menerapkan proses rekursi [5].

Komputasi bayangan juga dapat langsung dilakukan oleh *Ray Tracing*. Untuk menentukan apakah suatu titik pada obyek berada dalam bayangan atau tidak cukup dilakukan penembakan sinar dari titik tersebut ke sumber cahaya. Apabila sinar tesebut terhalang oleh obyek lain, maka titik tadi berada dalam bayangan.

Proyeksi perspektif juga langsung dilakukan oleh *Ray Tracing*. Pemanfaatan sifat sinar yang menyebar ke semua arah mengakibatkan perhitungan titik perpotongan sinar dengan obyek juga akan mengikuti prinsip-prinsip perspektif. Sehingga obyek yang jauh akan nampak lebih kecil. Secara logika dapat dijelaskan bahwa sinar akan lebih banyak memotong obyek yang lebih dekat dengan mata dari pada obyek yang lebih jauh. Dengan demikian titik perpotongan yang digunakan untuk membentuk obyek juga akan lebih banyak pada obyek yang lebih dekat. Hal ini mengakibatkan obyek yang lebih dekat akan terlihat lebih besar.

Ray Tracing memiliki beberapa keunggulan, diantaranya adalah [6]: a) Memiliki kemampuan memodelkan pencahayaan-pencahayaan secara presisi seperti pemantulan dan pembiasan dengan relatif mudah. Dengan kemampuan memodelkan pencahayaan ini, maka citra yang diperoleh juga semakin realistis. Model pencahayaan yang digunakan pun dapat diatur menjadi lebih kompleks. b) Memiliki kemampuan bekerja dengan baik dengan kelas obyek-obyek geometris yang lebih kaya daripada jaringan poligon. Obyek padat dapat dibentuk dari berbagai bentuk primitif geometris seperti bola, kerucut dan tabung. Bentuk-bentuk ini dimodelkan langsung dari persamaan matematisnya, sehingga diperoleh obyek yang benar-benar tepat. c) Beberapa elemen pada metode *render* lain dapat secara langsung diatasi oleh *Ray Tracing*. Elemen-elemen ini diantaranya adalah prinsip perspektif dan penghilangan permukaan yang tersembunyi.

Ray Tracing, seperti metode lainnya, juga memiliki beberapa kelemahan [6] antara lain: a) Waktu *render* yang relatif lama disebabkan karena komputer harus melakukan perhitungan pada setiap sinar untuk menentukan perpotongan sinar dengan obyek; b) Kebutuhan tenaga komputer yang cukup besar untuk melakukan komputasi menyebabkan metode *Ray Tracing* jarang digunakan untuk *render* secara *real time*, tetapi lebih banyak digunakan untuk me-*render* citra yang tidak langsung dibutuhkan hasilnya saat itu juga; c) Timbulnya *alias* pada citra yang dihasilkan apabila menggunakan metode *Ray Tracing* biasa.

Dari uraian di atas terlihat keunggulan dan kelemahan dari metode *Ray Tracing*. Tetapi, *Ray Tracing* tetap merupakan salah satu metode *render* yang layak untuk diterap-

Gambar 2: Citra tanpa anti-*aliasing*.

Gambar 3: Citra dengan anti-*aliasing*.

kan. Hal ini dikarenakan citra yang dihasilkan benar-benar realistis.

ALIAS DAN SAMPLING

Alias yang timbul dari proses *render* merupakan salah satu kelemahan dari metode *Ray Tracing*. *Alias* merupakan artefak yang pada citra tertentu akan nampak ketika tekstur di-*mapping* pada piksel. Hal ini diakibatkan oleh *sampling* yang kurang. *Sampling* adalah istilah dalam dunia grafika komputer untuk perhitungan warna tunggal untuk nilai satu piksel.

Alias yang timbul dalam *Ray Tracing* dapat diamati dengan timbulnya efek tangga atau yang biasa disebut *jaggies*. Efek *jaggies* biasanya tampak pada tepian obyek yang tidak vertikal atau horizontal. Tepi obyek tersebut menjadi kelihatan bergerigi (*jagged*).

Penyebab timbulnya efek tersebut adalah kenyataan bahwa hanya satu sinar yang diperhitungkan pada satu piksel. Padahal kenyataannya satu piksel tidaklah cukup kecil untuk merepresentasikan satu koordinat. Dengan kata lain efek ini muncul karena sinar pada satu piksel seharusnya lebih dari satu.

ANTI ALIASING

Untuk menghilangkan efek *jaggies* yang timbul pada citra hasil *render* maka dilakukan proses anti-*aliasing* [3]. Salah satu metode anti-*aliasing* adalah *super sampling*, yaitu menambah jumlah sampel. Seperti yang telah dijelaskan sebelumnya bahwa *alias* muncul karena proses *under sampling* maka dengan memperbanyak jumlah sampel *alias* bisa dihindari. Dengan kata lain, penambahan sampel memperbanyak jumlah sinar yang melewati satu piksel.

Memperbanyak sinar dapat pula dipandang sebagai proses memperbanyak piksel. Penambahan jumlah piksel berarti sinar yang ditembakkan pada tiap piksel juga semakin banyak. Hasil yang diperoleh dari proses *supersampling* ini kemudian di-*downsample* dengan cara menghitung rata-rata warna sub-piksel hasil *supersampling* menjadi warna satu piksel kembali. Jumlah sub-piksel yang dirata-rata tergantung berapa kali *sampling* yang kita tentukan untuk tiap piksel.

Sebagai gambaran, pada Gambar 2 dan Gambar 3 dapat dilihat citra yang tidak menggunakan dan menggunakan anti-*aliasing*. Pada Gambar 2 yang tidak menggunakan anti-*aliasing*, efek tangga tampak lebih tajam dibandingkan dengan Gambar 3 yang menggunakan anti-*aliasing*.

ORDERED GRID SUPER SAMPLING

Ordered Grid Super Sampling (OGSS) adalah suatu teknik untuk melakukan *render* pada resolusi lebih tinggi dan kemudian melakukan *downsample* dengan merata-rata warna sub-piksel disebut juga dengan *Ordered Grid Super Sampling* [1].

Sesuai dengan namanya piksel tersusun dalam bentuk grid. Sebagai gambaran, untuk empat kali *supersampling* maka proses yang dilakukan adalah melakukan *render* empat kali resolusi awal. Alternatif lain adalah melakukan *render* dua kali jumlah piksel dalam dimensi lebar dan dua kali jumlah piksel dalam dimensi panjang. Tiap piksel pada adegan akan direpresentasikan menjadi empat piksel. Setelah perhitungan pada sub-piksel dilakukan *downsample* atau pengambilan nilai rata-rata sehingga menghasilkan piksel yang telah di-anti-*alias*.

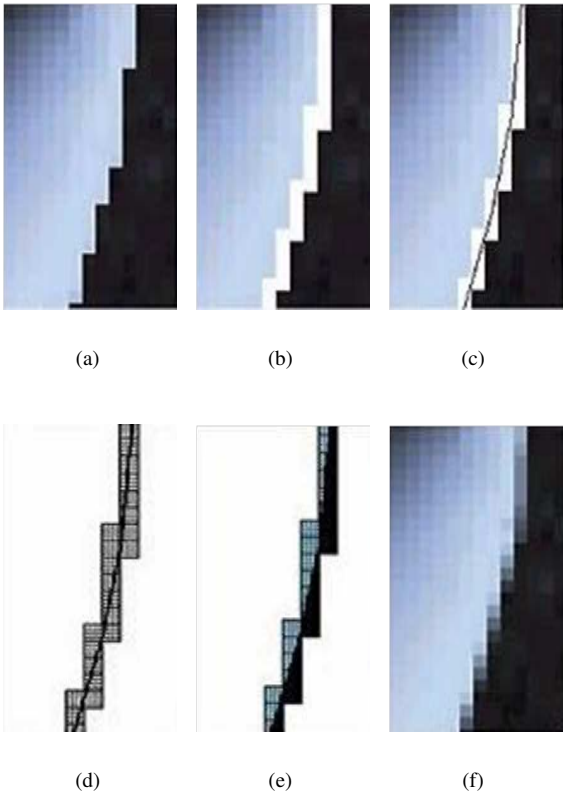
FRAGMENT ANTI ALIASING

Untuk melakukan anti-*aliasing* dengan teknik OGSS diperlukan sumber daya dan waktu yang besar pada saat komputasi. Hal ini disebabkan *sampling* dilakukan terhadap semua bagian dari adegan. *Sampling* dalam OGSS dilakukan secara menyeluruh. Piksel dalam adegan tidak semuanya mengalami *jaggies*, sehingga perlu dilakukan anti-*aliasing*. Hanya bagian tepian dari suatu obyek, atau bagian gambar yang urutan warna pikselnya mempunyai gradien yang perlu dilakukan anti-*aliasing*.

Fragment Anti Aliasing (FAA) adalah suatu metode baru yang digunakan untuk menutupi kelemahan pada teknik OGSS[2]. Pada teknik FAA ini, hanya bagian tertentu dari adegan yang dilakukan *sampling*. Bagian tersebut tentunya bagian yang mengalami efek tangga saja atau bagian dari adegan yang terdapat obyek.

Berikut adalah penggambaran proses *Fragment Anti Aliasing*: Gambar 4(a), adalah bagian yang mempunyai *jaggies* atau *alias* dari sebuah gambar. Gambar 4(b), FAA - 16x mengidentifikasi piksel non fragmen dari piksel fragmen. Piksel non fragmen kemudian ditulis dalam *frame buffer*. Gambar 4(c), Piksel fragmen dipisahkan dan kemudian ditulis dalam *fragment buffer*. Gambar 4(d), Fragmen diidentifikasi dengan melihat pojok-pojok segitiga dilakukan menggunakan ketelitian 16x (sub piksel 4 x 4). Gambar 4(e), dilakukan perhitungan dan ditemukan warna yang sesuai untuk bagian yang diperlukan. Warna disimpan dalam *fragment buffer*. Gambar 4(f) adalah hasil akhir setelah dilakukan anti-*aliasing*.

Agar bisa diketahui bagian yang perlu dilakukan *sam-*



Gambar 4: Citra dengan anti-*aliasing*.

pling, maka perlu dilakukan sebuah pengecekkan pada penelusuran setiap sinar. Dengan menerapkan FAA pada *render*, citra akan di-*render* dengan cara normal, tetapi saat didapati sebuah obyek, maka *render* dilanjutkan dengan *render* FAA [7].

Cara menentukan penggunaan proses FAA atau tetap melakukan *render* normal dimulai dengan menentukan jarak yang terjauh yang dapat dilihat dengan mata dan jarak terdekat yang tampak oleh mata. Setiap kali sebuah sinar ditembakkan maka dilakukan perhitungan untuk menentukan perpotongan antara persamaan sinar dan persamaan benda dalam citra. Setelah diketahui nilai koordinat perpotongan, maka diperiksa apakah titik perpotongan tersebut dalam jangkauan batas yang tampak pada mata. Apabila titik perpotongan tersebut benar atau valid, proses *render* dilakukan dengan menerapkan FAA sebagai berikut: a) Bangkitkan sinar; b) Temukan perpotongan sinar dan obyek; c) Apabila berpotongan, maka ((Y saat ini) -1) * variabel sampling. Proses tersebut dilakukan untuk memperbanyak jumlah piksel yang dilalui sinar. Demikian juga dengan ((X saat ini) - 1) * variabel sampling; d) Mengulang proses pembangkitan sinar, dan penelusuran sinar untuk nilai Y' dan nilai X' (yaitu nilai Y dan X yang telah diperbesar dengan perkalian variabel *sampling*); e) Dilakukan pengecekkan terhadap setiap sinar yang melewati x' dan y', untuk mengetahui apakah sinar-sinar tersebut masih berpotongan dengan benda. Apabila tidak terjadi perpotongan, proses *render* dengan FAA akan dihentikan dan dilanjutkan dengan *render* normal; f) *Render* normal akan mengambil nilai y' dan mendapatkan nilai y saat ini dengan membagi y' dengan variabel *sampling*.

Dari penjabaran di atas, dapat disimpulkan bahwa untuk menentukan nilai suatu warna untuk suatu piksel pada koordinat tertentu dengan metode FAA, piksel dalam koordinat tersebut akan diperbesar menjadi beberapa kali ukuran aslinya yang sesuai dengan besar variabel *sampling*.

IMPLEMENTASI

Penelusuran Cahaya

Pada *pseudo code render* gambar, tiap piksel pada bidang gambar akan dilakukan *plotting* warna. Warna tersebut dihasilkan dari perhitungan hasil penelusuran sinar sebagai berikut: dilakukan pemeriksaan apakah sinar mengenai benda dan ditentukan obyek terdekat yang terkena sinar paling awal; jika terdapat perpotongan dengan benda maka ditentukan vektor normal obyek tersebut; dilakukan penghitungan warna pada titik tersebut (titik yang terkena sinar); jika obyek mempunyai tingkat transparansi yang lebih besar daripada nol maka dilakukan penghitungan warna benda setelah mengalami transparansi.

Fungsi penelusuran cahaya akan mengambil warna pada titik tersebut, sedangkan apabila sinar tidak memotong sebuah obyek, warna yang akan dikembalikan oleh fungsi tersebut adalah warna latar belakang. Untuk lebih jelasnya proses ini dapat dilihat pada Gambar 5.

```
index <- FindClosestIntersection(Base, dir, Q)
  If index >= 0
     N <- DetermineNormal(obj(index), N, Q)
     if (Dot (N, dir) > 0)
            N.x <- -N.X
        N.Y <- -N.Y
            N.Z <- -N.Z
     ComputeLight(level, index, Q, N, dir, Iq)
            if obj(index).kt > TOL
               Transparan (level, index, Q, dir, Iq, obj(index))
     i.R <- Iq.r
     i.g <- Iq.g
     i.B <- Iq.b
     intersect <- True
  Else
     i.r <-BackGround.r
     i.g <- BackGround.g
     i.b <- BackGround.b
     intersect <- False
```

Gambar 5: *Pseudocode* penelusuran cahaya.

```
i.r <- objk.ka * objk.Ia.r
i.g <- objk.ka * objk.Ia.g
i.b <- objk.ka * objk.Ia.b
l.x <- theLight.LFw.x - Q.x
l.y <- theLight.LFw.y - Q.y
l.z <- theLight.LFw.z - Q.z

DiffScale <- Dot (L,L)
L <- Normalize(L)
DiffScale <- ScalaCahaya(Sqr(DiffScale))
If Inshow (Index, L, Q) <- False
  NdotL <- Dot(L,NormalN)
  if (NdotL > 0)
      i.r <- i.r + (objk.kd * objk.Ia.r * theLight.Lcolor.r * Ndot)/DiffScale)
      i.g <- i.g + (objk.kd * objk.Ia.g * theLight.Lcolor.g * Ndot)/DiffScale)
      i.b <_ i.b + (objk.kd * objk.Ia.b * theLight.Lcolor.b * Ndot)/DiffScale)
      CosAlpha <- -Dot(R, dir)
      if(CosAlpha > 0) Then
         i.r <- i.r + theLight.Lcolor.r + objk.ks + ((CosAlpha ^ objk.NO) / DiffScale)
         i.g <- i.g + theLight.Lcolor.g + objk.ks + ((CosAlpha ^ objk.NO) / DiffScale)
         i.b <- i.b + theLight.Lcolor.g + objk.ks + ((CosAlpha ^ objk.NO) / DiffScale)
         R.X <- -L.x + 2 * NdotL * NormalN.X
         R.Y <- -L.y + 2 * NdotL * NormalN.Y
 R.Z <- -L.z + 2 * NdotL * NormalN.Z
      if(level + 1) <= MAXLEVEL Then
         DotProd <- -Dot(dir, NormalN)
         R.x <- dir.x + 2 * DotProd * NormalN.x
         R.y <- dir.y + 2 * DotProd * NormalN.y
         R.z <- dir.z + 2 * DotProd * NormalN.z
         R <- Normalize(R)
         TraceTheRay(level + 1, Q, R, tempI)
         i.r <- i.r + tempI.r * objk.ks
         i.g <- i.g + tempI.g * objk.ks
         i.b <- i.b + tempI.b * objk.ks
      ComputeLight <- i
```

Gambar 6: *Pseudocode* penghitungan warna.

Perhitungan Warna pada Sebuah Titik

Pada proses perhitungan warna, terdapat beberapa faktor yang menentukan warna pada titik perpotongan, antara lain faktor *ambient* obyek dan jarak dari sumber cahaya, tempat sinar berasal.

Selanjutnya langkah-langkah untuk menentukan warna pada sebuah titik adalah sebagai berikut: a) melakukan pengecekkan, apakah benda terkena bayangan benda lain atau tidak; b) melakukan perhitungan *specular* terhadap benda; c) memasukkan komponen *diffuse* dari benda.
Proses penghitungan warna dapat dilihat pada Gambar 6.

Proses Anti-*Aliasing*

Untuk melakukan anti-*aliasing*, ada dua metode yang diimplementasikan yaitu:

OGSS, pada anti-*aliasing* dengan OGSS, proses untuk me-lakukan anti-*aliasing* dilakukan pada keseluruhan gambar. Tiap piksel akan diperbesar sesuai dengan jumlah *sampling* yang diinginkan. Setelah itu, proses *render* dijalankan, dan warna piksel hasil akhir adalah hasil rata-rata warna pada piksel sesuai jumlah *sampling*. *Pseudo code* untuk proses ini dapat dilihat pada Gambar 7.

FAA, citra dibentuk seperti *render* normal namun saat sinar ditelusuri pada awal penembakan dilakukan pengecekan pada sinar jikalau memotong obyek atau tidak. Apabila sinar memotong maka dilakukan anti-*aliasing*. Selanjutnya akan dilakukan pengecekkan apakah proses perbesaran *sampling* dilakukan terus atau kembali ke *render* normal apabila sudah tidak berpotongan dengan obyek.
Pseudo code proses FAA dapat dilihat pada Gambar 8, dan untuk proses *render*-nya dapat dilihat pada Gambar 9.

```
While Not (Y <- lastY)
f <- 0
    For X <- (ImWd / 2) * varsampling
        To((ImWd / 2) * varsampling) - 1 Step varsampling
    For h <- Y To Y - step Step-1
    u <- g * Uinc
    v <- h * Vinc
    dir.X <- centerV.X + u * RightV.X + v * UpV.X
    dir.Y <- centerV.Y + u * RightV.Y + v * UpV.Y
    dir.Z <- centerV.Z + u * RightV.Z + v * UpV.Z
    TraceTheRay(1, From, dir, i)
    If (i.R > 1) Then i.r <- 1
    If (i.g > 1) Then i.g <- 1
    If (i.B > 1) Then i.b <- 1
        rempCol(f) <- i
        thepix.r <- thepix.r / theray
        thepix.g <- thepix.g / theray
        thepix.b <- thepix.b / theray
        SetPixel ((g/varsampling) - 1 + (ImWd/2), (-h/varsampling) + (ImHt/2)),
         RGB(thepix.r * 255, thepix.g * 255, thepix.b * 255)
            f <- 0
    thepix.r <- 0
    thepix.g <- 0
    thepix.b <- 0
   Y <- Y - varsampling
```

Gambar 7: *Pseudocode* proses OGSS.

```
While Not (Y <- -(ImHt) / 2)
   For X <- -(ImWd / 2) To (ImWd/2)
            u <- X * aUinc
            v <- Y * aVinc
            dir.X <- centerV.X + u * RightV.X + v * UpV.X
            dir.Y <- centerV.Y + u * RightV.Y + v * UpV.Y
            dir.Z <- centerV.Z + u * RightV.Z + v * UpV.Z
            TraceTheFAARay(1, From, dir, i)
              if intersect <- True Then
                   ..... Render With FAA
            Else
            SetPixel (X + partX, -Y + partly),
              RGB(i.R * 255, i.g * 255, i.B * 255)
            Y <- Y - 1
```

Gambar 8: *Pseudocode* proses FAA.

Tabel 1: Euclidean distance antara OGSS-FAA.

OGSS-FAA	Persentase OGSS-FAA
12.4485	4.881764706
7.6969	3.018392157
1.9062	0.747529412
9.5357	3.739490196
4.4707	1.753215686
23.9536	9.393568627
11.5808	4.541372549
13.7475	5.391176471
3.8944	1.527215686
12.7145	4.986078431

UJI COBA

Uji coba dilakukan untuk mengetahui kemampuan perangkat lunak dalam me-*render* adegan. Tolok ukur yang dipakai dalam uji coba adalah waktu dan kualitas citra yang dihasilkan. Uji coba dilakukan dengan mengubah-ubah komponen yang mempengaruhi *render* dalam adegan, antara lain: a) Besar *sampling*, yaitu perbesaran yang dilakukan untuk melakukan anti-*aliasing* (lihat Gambar 10 dan Gambar 11); b) Resolusi gambar (lihat Gambar 12); c) Jumlah obyek (lihat Gambar 13).

Pada uji jarak *Euclidean* untuk menganalisis citra yang dihasilkan proses *render* baik menggunakan OGSS maupun FAA, maka dilakukan perhitungan jarak *Euclidean* yang dinyatakan dengan Persamaan (1).

$$\frac{1}{XY}\sum_{i=1}^{x}\sum_{j=0}^{y}\sqrt{(R_{ij}-R'_{ij})^2}+\ldots \\ \ldots(G_{ij}-G'_{ij})^2+(B_{ij}-B'_{ij}) \quad (1)$$

Dari persamaan tersebut maka dibentuk citra yang berbeda-beda dan dianalisis sehingga diperoleh nilai-nilai seperti pada Tabel 1. Analisis dilakukan dengan menghitung jarak *Euclidean* antara citra *render* OGSS, dengan citra *render* FAA.

Dari perhitungan citra hasil OGSS dan FAA maka didapatkan nilai persentase jarak *Euclidian* antara keduanya kurang dari 6%. Hal ini berarti citra yang dihasilkan oleh proses *render* FAA dan citra oleh proses OGSS relatif sama.

SIMPULAN

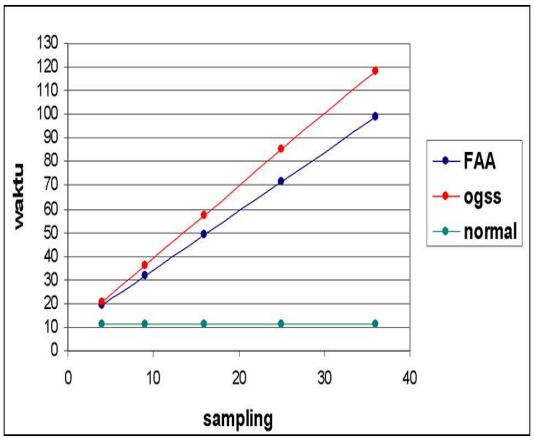
Dari data dan analisis yang dilakukan, dapat ditarik simpulan sebagai berikut: a) Pada lebar gambar yang sama dan dengan jumlah *sampling* yang berbeda-beda maka semakin besar *sampling* untuk melakukan anti-*aliasing*, maka semakin besar waktu yang diperlukan untuk me-*render* adegan; b) Pada *render* dengan FAA, semakin besar *sampling* maka waktu *render* akan semakin besar pula, tetapi masih lebih cepat dibandingkan OGSS; c) Pada besar *sampling* yang sama tetapi dilakukan dengan resolusi gambar yang

```
Fy <- (Y + 1) * varsampling
tempX <- -ImWd / 2 * varsampling
lastY <- -ImHt / 2 * varsampling
While Not Fy <- lastY
          if ky <- 3 Then
                Exit Do
          realliHit <- False
For Fx <- tempX To ImWD / 2 * varsampling Step varsampling
    for h <- Fy To Fy - step Step-1
        For g <- Fx To (Fx + step)
            u <- g * Uinc
            v <- h * Vinc
            dir.X <- centerV.X + u * RightV.X + v * UpV.X
            dir.Y <- centerV.Y + u * RightV.Y + v * UpV.Y
            dir.Z <- centerV.Z + u * RightV.Z + v * UpV.Z
           TraceTheRay(1, From, dir, i)
            If intersect <- True Then realliHit <- True
            If (i.R > 1) Then i.R <- 1
            If (i.g > 1) Then i.g <- 1
            If (i.B > 1) Then i.B <- 1
            rempCol(f) <- i
              thepix.R <- (rempCol(f).R + thepix.R)
              thepix.g <- (rempCol(f).g + thepix.g)
              thepix.B <- (rempCol(f).B + thepix.B)
              f <- f + 1
              thepix.R <- thepix.R / theray
              thepix.g <- thepix.g / theray
              thepix.B <- thepix.B / theray
            SetPixel((g / varsampling) - 1 + partX, (-h/varsampling) + partY),
             RGB(thepix.R + 255, thepix.g * 255)
            If realliHit <- False Then
            ky <- ky + 1
            thepix.R <- 0
            thepix.g <- 0
            thepix.B <- 0
            Fy <- Fy - varsampling
      Y <- ((Fy) / varsampling)
      X <- (ImWd / 2)
```

Gambar 9: *Pseudocode* untuk *render* pada FAA.

berbeda-beda, terlihat kinerja *render* adegan dengan menggunakan FAA masih lebih baik daripada OGSS; d) FAA dan OGSS menghasilkan citra yang relatif sama yang didukung oleh analisis jarak dengan perhitungan jarak *Euclidean*; e) FAA akan lebih sesuai untuk gambar dengan ukuran besar, tetapi dengan jumlah obyek sedikit atau kecil, sedangkan OGSS digunakan untuk gambar ukuran kecil dengan jumlah obyek yang banyak.

Beberapa pengembangan yang dapat dilakukan selanjutnya adalah: a) Mengembangkan editor adegan, sehingga pengguna lebih mudah dalam mendefinisikan *render* terhadap adegan yang diinginkan; b) Menambah kompleksitas untuk meningkatkan kualitas dari *Ray Tracing* itu sendiri dan melakukan optimasi dalam *render* dengan *Ray Tracing*; c) Menambahkan jenis-jenis obyek yang tidak hanya bentuk geometris tetapi bentuk-bentuk bangun yang dibangun dengan poligon hasil pendefinisian dari pengguna.



Gambar 10: Grafik hasil uji coba *sampling* terhadap waktu pada resolusi 600 x 600.

DAFTAR PUSTAKA

[1] Watt, Alan: *3D Computer Graphics*. 3rd edn. Addison-Wesley (2000)

[2] Baron, Dave.: *How It Works: Fragment Anti Aliasing*. http://firingsquad.gamers.com/guides/fragmentaa/page2.asp [diakses pada 4 Des 2002]

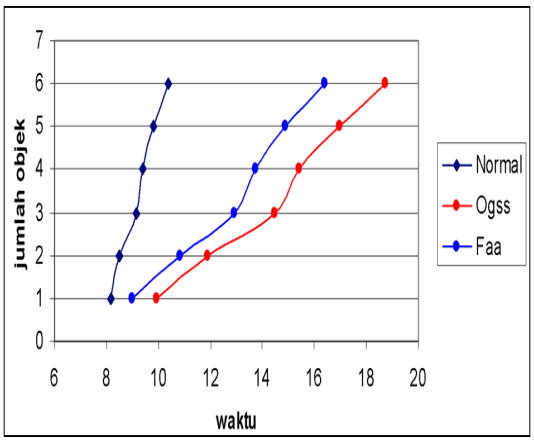
[3] Heam, Donald, Pauline Bake: *Computer Graphics*. 2nd edn. Prentice Hall Inc. (1994)

[4] Chaudhuri, Siddhartha: *FuzzyPhoton*. http://fuzzyphoton.tripod.com (2002)

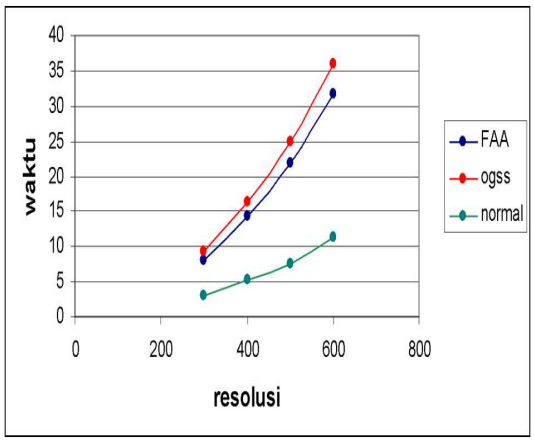
[5] Buck, Jamis: *The Recursive Ray Tracing Algorithm*. http://www.geocities.com/jamisbuck/TheRecursiveRayTracingAlgorithm.htm [diakses pada 25 Apr 2000]

[6] Lu, Charity, Alex Roetter, Amy Schult: *Ray Tracing*. http://cse.stanford.edu/class/sophomore-college/projects-97/ray-tracing.html [diakses pada 2002]

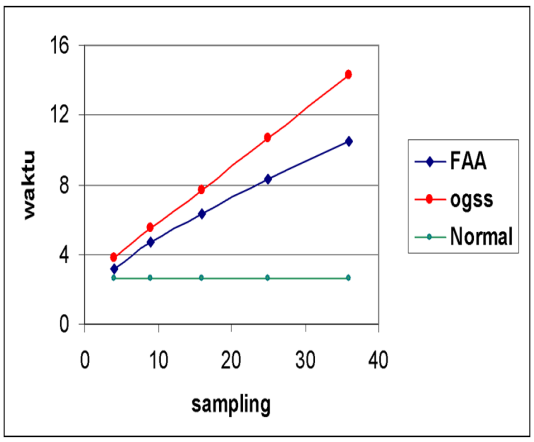
[7] Weinand, Lark: *Matrox Parhelia - 512 - The Challenger Fragment Anti Aliasing*. http://www6.tomshardware.com/



Gambar 13: Grafik hasil uji coba keterkaitan jumlah obyek dan waktu *render*.



Gambar 12: Grafik hasil uji coba resolusi 200 - 800 terhadap waktu.



Gambar 11: Grafik hasil uji coba *sampling* terhadap waktu pada resolusi 300 x 300.

graphic/20020514/parhelia-06.html
[diakses pada 14 Mei 2002]